Can hierarchical clustering improve the efficiency of non-linear dimension reduction with spring embedding?

Michael Schroeder and George Katopodis
Department of Computing, City University
Northampton Square, London EC1V 0HB, UK
{msch,dc707}@soi.city.ac.uk

July 4, 2002

Abstract

In visual datamining proximity data, which encodes the relationship between some entities as a distance, is often available. This proximity data is inherently high-dimensional and can be mapped into a low-dimensional 2D or 3D target space such that the points in the target space adhere to the specified distances. The target space can then be visualised as scatterplot.

Force-directed graph drawing such as spring embedding can be used for this purpose and produce non-linear mappings. Spring embedding starts with a random layout, which is continually improved. This process is time-consuming since the basic algorithms perform in $O(n^3)$, where n is the number of entities.

In this paper, we consider hierarchical clustering, which performs in $O(n^2)$ as pre-processing for spring embedding. The clustering information is used to generate an approximate initial layout for the spring embedding algorithm. We compare this clustering-based initial layout with a number of others regarding their complexity, speed-up, and convergence behaviour.

Keywords: Dimension reduction, multi-dimensional scaling, force-directed graph drawing, spring embedding, hierarchical clustering

1 Introduction

Recently there has been increased interest in visual datamining, which marries the three areas of information visualisation, datamining, and human-computer interaction. Visual datamining is a process in which data is analysed, processed, and then visualised. Often the data defines pair-wise distances between the entities under consideration. For such proximity data, there are a number of analysis techniques, such as hierarchical clustering and multi-dimensional scaling. Hierarchical clustering [Gor81, Eve78, Kru77, DEKM98, Web99] aims to cluster similar objects together. As the clustering is hierarchical, this processing technique is naturally visualized in the next phase as a tree. Multi-dimensional scaling (see e.g. [Gor81, Eve78, Kru77, MKB79, Web99]), on the other hand, maps inherently high-dimensional proximity data into a 2D or 3D target space, such that distances between the entities in the target space reflect their distance as given in the proximity data. The target space can then be visualised as a scatterplot.

Proximity data can be seen as a label, fully connected, undirected graph and as a result graph drawing techniques can be useful for multi-dimensional scaling. One such graph drawing method is spring embedding [Kru64, QB79, KK89, FR91,

AAH94, RD96, Tun99, BETT99]. Spring embedding is a local optimisation technique, which starts with an initial (usually random) layout and then iteratively improves this layout by viewing edges between nodes as springs, thus leading to attractive and repulsive forces based on the desired distance between the nodes. Nodes move in the direction of this force until a state of minimal energy is reached.

In this paper, we investigate the hypothesis whether hierarchical clustering can used as pre-processing to improve the efficiency and quality of spring embedding. The paper is orgranised as follows: First, we give a motivating example from biology and show dendrograms resulting from hierarchical clustering and a scatterplot resulting from spring embedding produced with our Space Explorer tool [SGvHN01]. Next, we give an overview over hierarchical clustering, multi-dimensional scaling, and spring embedding. In section 3, set the scene to test our hypothesis by introducing six approaches to an initial layout. In section 4, we carry out experiments and discuss the results. Next, we discuss in section 5 another highly important parameter for an efficient layout besides the initial layout. Finally, we conclude with some comparison and conclusions.

2 Visual datamining

Before we present a short summary of hierarchical clustering and a more detailed background of spring embedding, let us consider an example.

2.1 Motivating Example

Advances in molecular and structural biology have resulted in a great output of biological data. The advent of DNA chip technology [dIB97] allows to measure systematically the level of expression of several thousands of genes. Such gene expression can represented as table (see Fig. 1). We selected an example from an analysis of the cell cycle in yeast [ESBB98]. The authors selected 800 genes whose level of expression fluctuates periodically during the cell cycle. Each gene is characterized by a series of expression measurements taken at successive time intervals. The alpha cell experiment contains 18 successive measurements on the same cell population. We are thus dealing with multivariate analysis, with a data matrix of 800 rows (entries) and 18 columns (variables). Next we translate the multivariate data to proximity data. Many distance metrics can be used to this end.

Definition 1 *Let $x, y \in \mathcal{R}^n$. Then*

$$d_a(x,y) = acos(\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}})$$

is called angular separation and

$$d_e(x,y) = \sqrt{\sum_{i=1}(x_i - y_i)^2}$$

os called Euclidean distance.

For this example, we will use angular separation as it is a good choice to compare similar behaviour of genes, rather than their absolute expression levels.

How can we analyse the above data? Hierarchical clustering and spring embedding are two techniques to visualuie the data as dendrograms and scatterplot respectively.

ORF	α_0	α_7	$\alpha_1 4$	...
YER150W	0.41	1.47	1.8	...
YGR146C	0.78	0.37	-0.09	...
YDR461W	2.36	2.35	2.3	...
...				

$\Longrightarrow$

ORF	YER 150W	YGR 146C	YDR 461W	...
YER150W	0	0.3	0.6	...
YGR146C		0	0.4	...
YDR461W			0	...
...				0

Figure 1: Left: Fragment of a multivariate data table. Rows correspond to genes, and columns to experiments. The level of expression of ca. 6000 genes was measured at 28 successive time points [ESBB98]. Using for example angular separation, the multi-variate data is converted to proximity data (table on right).

	a	b	c	d
a	0	1	6	8
b		0	5	7
c			0	2
d				0

$\Longrightarrow$

	(a,b)	c	d
(a,b)	0	5	7
c		0	2
d			0

$\Longrightarrow$

	(a,b)	(c,d)
(a,b)	0	5
(c,d)		0

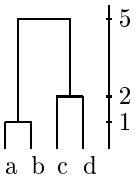


Figure 2: Hierarchical clustering using single linkage and the resulting dendrogram below.

2.2 Hierarchical Clustering

Hierarchical clustering identifies similar objects and clusters them together.

It consists in separating a set of objects into several subsets on the basis of their similarities. This problem is not simple, and various approaches have been developed to optimise the clustering in function of some criteria. The underlying idea is generally to define clusters that minimise intra-cluster variability while maximising inter-cluster distances. One particular approach is hierarchical clustering [Gor81, Eve78, Kru77, DEKM98, Web99]. Consider Figure 2. The first step is to identify the two objects with the closest relatedness. This pair builds the first cluster. The next pair is then searched, but this time a pair can be formed by joining either two objects, or an object and the cluster. The process is then repeated iteratively, by forming a cluster from the pair of closest objects/clusters, until all of them are connected forming a rooted tree. The clustering process requires to define distance not only between objects, but also between an object and a cluster or between two clusters. Several options can be used for this such as e.g. minimal, maximal or average distance (single, complete, or means/UPGMA (unweighted pair group method using arithmetic averages) [SM58] linkage respectively (see [Gor81, Eve78, Web99])). This choice has an impact on the cluster formation, and may lead to different interpretations (see e.g. [Gor81, Eve78, Kru77, Web99]). In general the above clustering algorithm runs in $O(n^2)$, where n is the number of entities. However, for the single linkage or nearest neighbour method $O(n^2)$ can be achieved [Ols95]. Applied to the above data set, one obtains the dendrogram in Fig. 3, which has ben additionally enriched by a colour map.

2.3 Spring Embedding

Before we explain how to use spring embedding as a non-linear multi-dimensional scaling technique, let us review spring embedding.

Spring embedding [Kru64, QB79, KK89, FR91, AAH94, RD96, Tun99, BETT99] uses the physical metaphor of springs (see Fig. 4). According to Hooke's law, there are attractive and repulsive forces based on the desired distance between the nodes. The Spring embedding algorithm starts with a random layout and then computes

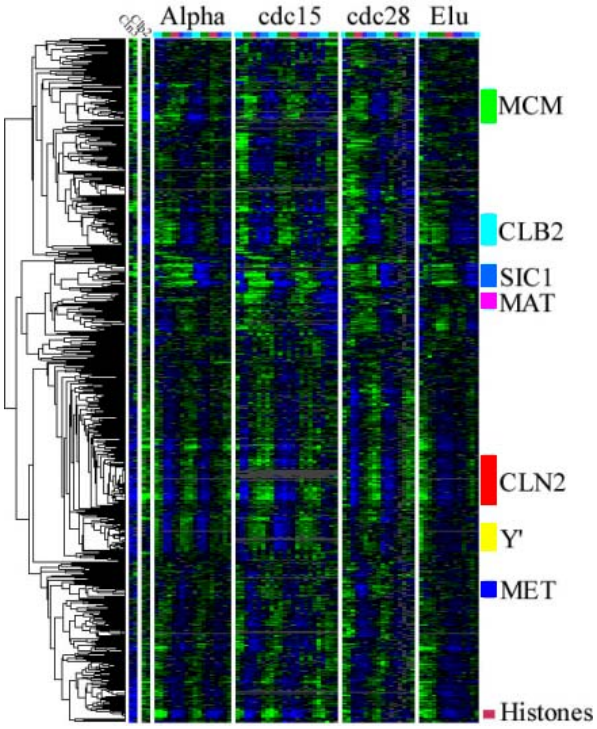


Figure 3: Dendrogram visualisation [ESBB98] for gene expression data.

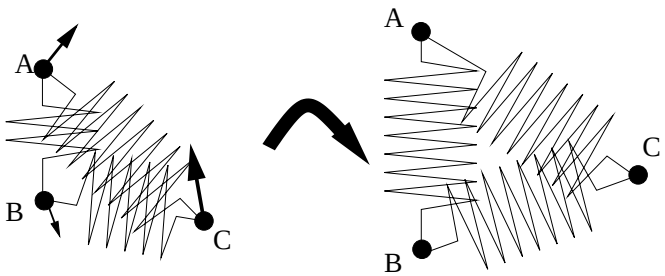


Figure 4: Multi-dimensional scaling with spring embedding. Nodes are moved into the direction of the sum of forces.

for a number of iterations the forces for each object and moves it into the direction of the overall force until a state of minimal energy is reached. Nodes are usually not moved by the full amount of the force, but are limited by a maximum amount known as temperature. In a technique known as temperature scheduling the temperature is reduced per iteration, so nodes are more and more limited in their movement.

An early example of a force directed drawing algorithm is Tutte's Barycenter method [Tut60, Tut63], in which the force acting on a node i is given as $F(i) = \sum_{i=1,i\neq j}^{n} x_i - x_j$, where x_i are the coordinates of node i. The method starts by placing a set of at least three fixed vertices in positions where spring forces cannot affect them. The rest of the vertices are free and initially located at the origin. The algorithm iterates and in each iteration a new location is computed for each free vertex.

Since then a number of different algorithms have been developed. An influential algorithm extended later in various ways is Eades' algorithm [Ead84]. Besides springs of logarithmic strength governed by Hooke's law it views nodes and non-adjacent vertices as electrically charged particles repelling each other.

Fruchterman and Reingold's FR algorithm [FR91] extends Eades' algorithm by introducing new attractive and repulsive factors as well as the above mentioned idea of temperature and temperature scheduling. Their algorithm computes all forces on all nodes, which are then moved at once. The algorithm limits the maximum displacement to a temperature, which starts at an initial value and decreases in an inverse linear fashion. They note that a better cooling schedule can dramatically change resulting layouts and reduce the number of iterations required to find the layout. To speed up their algorithm they use what they call grid-variant algorithm. In this variant the drawing area is divided into grids. The attractive forces are computed as usual, but for each vertex the repulsive force is only computed between the vertex and the nodes in its close-by cells of the grid. This method has not been very popular mainly due to its dependency on the distribution of nodes within the grid, the overhead of placing nodes in grid cells in each iteration, and the compromised layout quality that it produces in some cases. FR terminates after a maximum number of iterations is exhausted. Since different types of graphs require different number of iterations for their layouts to converge, this method of stopping may terminate too early or too late depending on the graph.

GEM [AAH94], a short form for Graph EMbedder, is a Spring algorithm based on Eades' algorithm, and is the first one that uses the history of a node's movement to choose the temperature for the current displacement of the node. Unlike FR, this algorithm moves the nodes one at a time in each iteration. The algorithm iterates until a maximum number of iterations is carried out or until the average local temperature of all nodes falls below a threshold. It also computes a likelihood that a node is oscillating or rotating and changes its temperature accordingly. GEM also starts with an incrementally computed initial layout, which is particularly benefitial for trees and grids.

Kamada and Kawai's KK algorithm [KK89] is based on graph-theoretic distances between pairs of vertices. In KK, the graph nodes are considered as particles that are all connected by springs whose ideal lengths are equal to the graph-theoretic distances between their two end-point particles multiplied by the desirable length of one edge. The goal of the algorithm is to find a balanced spring system. The major drawback of this method is its high computational cost as partial differential equations need to be solved.

Tunkelang [Tun94] has presented an incremental algorithm with three stages. In the first stage, a permutation of the nodes of a given graph is constructed from a minimal height breadth-first spanning tree of the graph. In the second stage, for each node in the order computed in the first stage, the area surrounding the already positioned neighbours of the node is sampled and examined for an approximate best

position, and the node is placed at this position. In the second stage, when the local optimisation procedure improves a vertex's position, it recursively performs the process on the already placed adjacent nodes of the vertex. Finally in the last stage, the algorithm performs the local optimisation process at every node for fine tuning.

This algorithm generates layouts that are different from those of FR, KK, and GEM [FMC96]. Specifically, it does not capture graph symmetries. In addition, the algorithm takes a quality parameter where a large value results in better quality of graph layouts. However, this quality parameter is estimated to have an exponential effect on the running-time of the algorithm [FMC96].

Tunkelang proposed a second algorithm [Tun99] based on the FR algorithm. This algorithm differs from FR in its computation of repulsive forces, in its optimization process, and in stopping the algorithm. Similar to the "grid variant" in FR, the algorithm approximates the computation of repulsive forces on a node. FR's optimization process uses force laws that in effect compute the negative gradient of an implicit objective function. However, this algorithm uses the conjugate gradient method on a non-quadratic objective function using an approximate line search. In addition, this algorithm uses the average of the square of the displacement distances of nodes for stopping the algorithm. It terminates when this value is less than 0.01.

Davidson and Harel's DH algroithm [RD96] uses simulated annealing. In each step, the layout is improved by comparing the current position of a node to one randomly selected with the neighbourhood of the node. Step by step, the temperature is reduced and the neighbourhood gets smaller. The comparison of current to randomly selected position involves weighted factors for a number of criteria such as overall stress, edge crossings, etc.

All the above algorithms reach their goals and produce aesthetically pleasing drawings. FR, KK, GEM and DH often produce similar drawings and they display symmetry. Tunkelang's first algorithm [Tun94] often yields different drawings without symmetry. DH is the most flexible but also the most time consuming while GEM and KK are very competitive in speed. Finally, FR is fast on small graphs, but slows down on graphs with more than 60 nodes [FMC96].

How can we use spring embedding to analyse the proximity data discussed earlier? I.e. how can spring embedding implement multi-dimensional scaling?

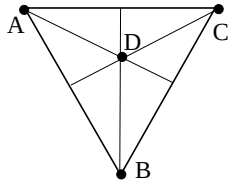
2.4 Multidimensional scaling with Spring Embedding: from distances to coordinates

Multi-dimensional scaling [Sam69, Gor81, Eve78, Kru77, MKB79, Web99] aims to find points in space which satisfy some given distances. Formally the problem can be defined as follows:

Problem: We have to define an algorithm which computes a matrix $X = (x_1, \ldots, x_n) \in \mathcal{R}^{m,n}$ for a proximity matrix $D = (d_{ij}) \in \mathcal{R}^{n,n}$ such that $d_e(x_i, x_j) = d_{ij}$, i.e. the Euclidean distance $d_e()$ between x_i and x_j is d_{ij}.

Before we show how to construct a solution for problem, let us note a limitation. Given a proximity matrix it may not be possible to find a solution at all.

Not every distance matrix can be visualized in Euclidean space. Consider e.g. the Figure below, where A, B, and C have all the same distance and D is in the middle of the triangle formed. As it stands the drawing is Euclidean (otherwise we could not draw the figure!). But now consider a distance matrix, where A, B, C are all equidistant, but the distance of A, B, and C to D is slightly less than in the drawing on the right.



Such a distance matrix satisfies all of the requirements of a metric such as triangle

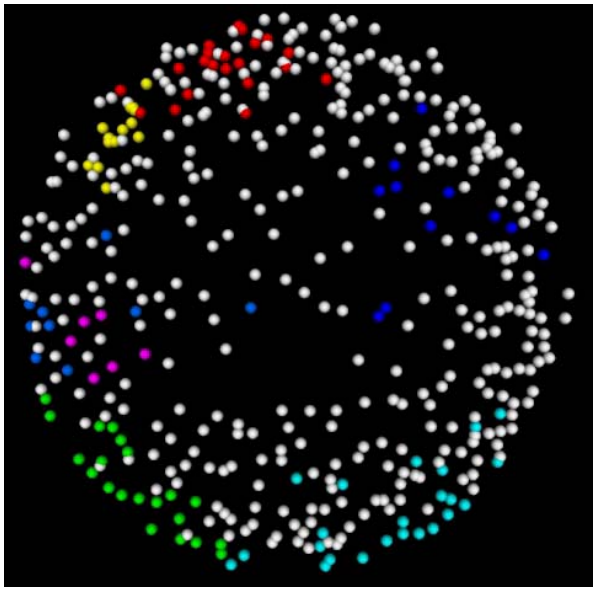


Figure 5: Scatterplot of gene expression data produced with spring embedding and anguar separation using SpaceExplorer [SGvHN01].

inequality, yet it is impossible to draw these distances in any Euclidean space.

In general, there are two approaches to multi-dimensional scaling. The mapping can be linear or non-linear. Principal coordinate analysis (PCoA) [Gow66] is an example for a linear mapping, while spring embedding is one for a non-linear mapping. How does it work?

Graph drawing techniques are applicable to proximity data, as the proximity data can be seen as specification of a complete, labeled undirected graph. There is however a difference: While graph drawing techniques as discussed above emphasise aesthetic aspects [BETT99], this is not important for proximity data as the low-dimensional drawings can rarely meet the given distances, which are inherently high-dimensional. Hence, aesthetics is not of importance, but rather to achieve a layout with minimal energy. As an example for a scatterplot of the gene expression data introdcued in 2.1 consider Fig. 5.

A typical force directed algorithm, which is used for the experiments in this paper, looks as follows:

Algorithm Given a distance matrix D.

1. Set all nodes to an initial position, i.e. $X = (x_1, \ldots, x_n) \in \mathcal{R}^{m,n}$ is populated with random values.

2. Loop: For all $1 \leq i \leq n$ compute the sum of attractive and repulsive forces $\Delta x_i = F_i^{attr} + F_i^{rep}$, where

$$\begin{aligned} F_i^{attr} &= \sum_{j=1}^{n} \frac{d_{ij} - d_e(x_i, x_j)}{3 d_e(x_i, x_j)} (x_i - x_j) \\ F_i^{rep'} &= \sum_{j=1}^{n} \frac{x_i - x_j}{d_e(x_i, x_j)} \\ F_i^{rep} &= 2F_i^{rep'} / d_e(F_i^{rep'}, 0) \end{aligned}$$

3. For all $1 \leq i \leq n, 1 \leq h \leq m$ let $x_{hi} := x_{hi} + max\{-5, min\{\Delta x_{hi}, 5\}\}$, i.e. adjust x_{hi} by Δx_{hi} but limit the change by a "temperature", 5 in this case.

The complexity of this naive algorithm is $O(m * n^2)$, where m is the number of iterations of the main loop and n is the number of nodes. Typically m will depend on n. If we choose a linear dependency then the overall algorithm performs in $O(n^3)$.

3 Improving the efficiency of Spring Embedding

Two parameters in the algorithm above influence its convergence to a solution: First, the initial layout and second, the temperature. In this section, we want to explore the former. We leave the temperature fixed and investigate how different initial layouts influence the convergence behaviour of the algorithm. Besides improving the convergence our main focus will be on the complexity of generating the initial layouts. Obviously, the pre-processing should not worsen the spring embedding algorithms performance.

We compared six different initial layouts:

- Random. A uniform distribution is used to place the nodes randomly
- Zero. All nodes are initially placed at 0.
- Circle. All nodes are initially place on a circle. Two subsequent nodes have the same angle between each other. The circle radius is the maximum distance found in the proximity data.
- Clustering. Hierarchical clustering is applied to the data and the clusters are mapped to coordinates.

  Consider the dendrogram in Figure 2. Besides displaying the clusters, the horizontal lines in the dendrogram convey the distances between nodes and between clusters. Instead of using the one dimension only, we can use the second dimension as well and use such a drawing as initial layout. Consider the algorithm in Fig. 6. The mapping from clusters to coordinates uses three parameters: the cluster root, the current position in the drawing and the current direction, which can be vertical or horizontal. We traverse the tree top-down. If the root is a leave, we can place it at the current position. Otherwise, we need the distance d between the two children clusters. Next we call the procedure recursively for each child with updated position and direction. If the direction was horizontal it is set to vertical and vice versa. Depending on the direction, the position's x or y coordinate are set to $+/-\frac{d}{2}$. As an example, consider Fig. 7.
- Spanning tree. Nodes are place on a line. We start with a randomly chosen node placed at 0. For the previous node n placed at x, the nearest neighbour m, which has not yet been placed on the line, is selected. If d is the required distance between n and m, then m is placed at $x + d$.
- PCoA. As described above, Principal Coordinate Analysis (PCoA) [Gow66] is a multi-dimensional scaling technique, which computes a linear mapping. Thus, it is a competitor to spring embedding, which computes a non-linear mapping. But of course we can combine the two and let spring embedding try to improve the linear mapping.

Figure 8 shows the complexity of these Pre-processing techniques. Theoretically, all of them are useful, since they are not worse then spring embedding itself. Practically, however, there exist faster spring embedding algorithms, so that we pre-processing should not be worse than $O(n^2)$.

```
cluster2coord(root, dir, x,y)
 if root is leave then place root at x,y
 else
   let d be distance between the root's two children
   if dir is horizontal then
     cluster2coord(left child,  vertical, x-d/2, y)
     cluster2coord(right child, vertical, x+d/2, y)
   else
     cluster2coord(left child,  horizontal, x, y-d/2)
     cluster2coord(right child, horizontal, x, y+d/2)
   fi
 fi
```

Figure 6: Algorithm to map hierarchical clusters to coordinates.

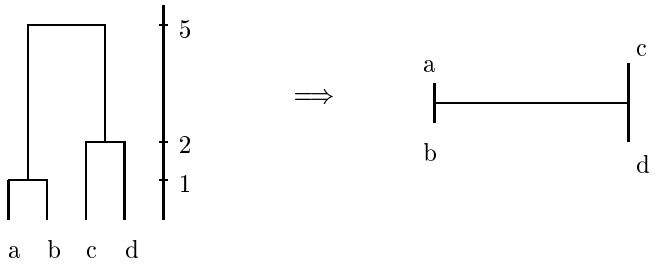


Figure 7: Example of mapping hierarchical clusters to coordinates.

Initial Layout	Random	Zero	Circle	Clustering	Spanning tree	PCoA
Complexity	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^3)$

Figure 8: Complexity of the pre-processing methods used, where n is the number of nodes.

4 Experiments and Results

Let us evaluate the above pre-pocessing methods: We use three data sets.

- The data in Fig. 9 contains pair-wise distances for 72 European cities. This data is not inherently high-dimensional and therefore can be laid out in 2D without large errors. Thus the stress of the spring embedding layout should converge to a low value.
- The data in Fig. 10, left contains random distances, which were derived from uniformly distributed nodes. This is an example of data, where dimension reduction is very limited, i.e. the stress of any layout in 2D will be high. The uniform distribution also means that the data has no clusters, which are the very basis for methods like the spanning tree and clustering method.
- The data in Fig. 10, right contains random data, which is hierarchically normal distributed. We decided on a number of clusters and created for each a point as cluster center using a large mean and variance. Then we recursively, created subclusters around those cluster centers. Thus the data has exactly the structure hierarchical clustering is assuming.

Consider Fig. 9 and the six initial layout described above. As argued above, the data can be laid out in 2D and hence the stress of all techniques approaches 0. However, notably, PCoA, which is linear mapping, produces a very good layout, which cannot be improved by spring embedding. The circle layout performs worst of all methods followed by the spanning tree method, which did not produce a very good initial layout. In fact, the initial random layout and the zero layout, which behave both the same, start with a 3 times better initial layout than spanning tree. Most interestingly, clustering performs very well - better than random. Thus our hypothesis that clustering can be beneficial as initial layout is supported by this data set. But let's look at others.

Consider Fig. 10, left. Since the circular layout performed so bad, it is not included in the figure. As argued above, this data set does not contain structure and thus spanning tree and clustering do not pay off. Surprisingly, the random layout performs very well and even better than PCoA. Thus, we have to refine our hypothesis: clustering can improve spring embedding, but only for data sets, where it is applicable. But let us see how good clustering can get, if the data contains hierarchical clusters.

The third data set was designed for this purpose. Consider Fig. 10, right. Clustering performs best of all methods and starts with a layout about 4 times better than the random one.

As an overall conclusion, we can say that hierarchical clustering, whose complexity is lower than spring embedding, can improve spring embedding, but only if the data it is applied to contains structure and clusters.

5 Temperature Scheduling

As explained in 3, besides the initial layout, temperature scheduling plays an important role in the convergence of the layout quality. In the above experiments, we used a fixed temperature. Let us now look into how this can be improved. Consider Fig. 11. Instead of a fixed temperature, it defines the temperature as a percentage of the force on the node. The figure shows that an optimal percentage is reached at 6%, beyond this point (e.g. at 8%), the stress oscillates. The percentage is not fixed, but as can be seen in Fig. 11, right, the oscillation threshold depends on the

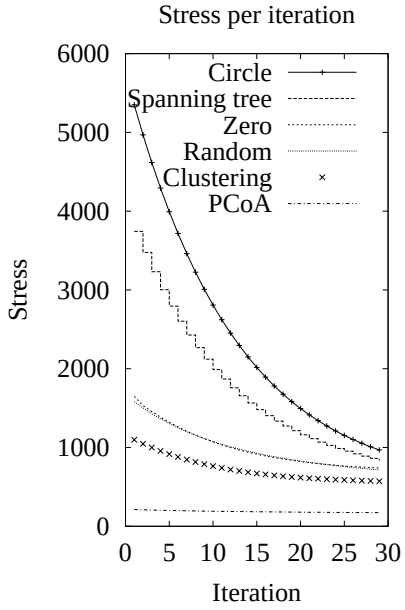


Figure 9: Stress/Iterations for different initial layout applied to pairwise distances of 72 European cities.

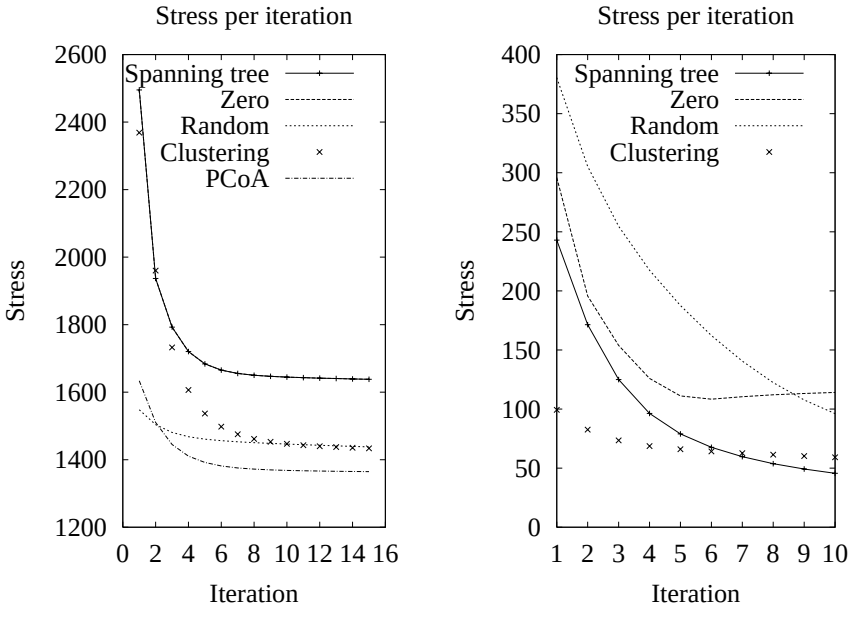


Figure 10: Stress/Iterations for different initial layouts. Left: distances calculated for nodes uniformly distributed. Right: distances for nodes hierarchically normal distributed.

number of nodes. The more nodes there are, the larger the forces acting on a node are and the smaller the force percentage used for movement of the nodes should be.

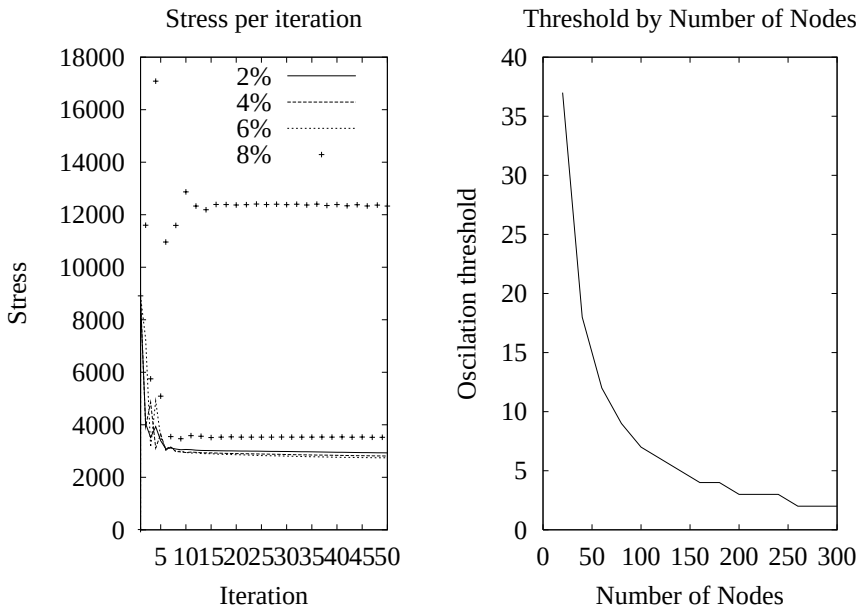


Figure 11: Left: Stress/Iteration for temperature as percentage of force. 6% leads to the best behaviour and beyond this (8%) the stress oscillates. Right: Number of nodes/oscillation threshold. Depending on the size of the data set, a different percentage is best.

Overall the temperature can highly influence the convergence behaviour of the stress function. Fig. 12 shows the stress functions for fixed temperature and for the temperature as percentage of the force. The latter performs clearly better.

6 Comparison and Conclusions

Visual datamining is a process of data preparation, processing, and visualisation. Often the data is inherently high-dimensional proximity data, which can be mapped to 2D or 3D and then visualised as scatterplot. One such mapping - a non-linear one - can be realised by force directed graph-drawing with spring embedding [Kru64, QB79, KK89, FR91, AAH94, RD96, Tun99, BETT99]. In this context of multi-dimensional scaling [Gor81, Eve78, Kru77, MKB79, Web99], spring embedding has advantages over other methods such as PCoA [Gow66] as it is a flexible anytime-algorithm which comes up with an approximation of a solution with any time limitations. But similar to PCoA, the basic spring embedding algorithm has cubic runtime in the number of nodes, if one assumes that the outer loop for the force computations depends linearly on the number of nodes. One way in which spring embedding's performance can be improved is to use pre-processing to start with an initial layout, which already approximates a solution instead of a purely random initial layout. The hypothesis we investigated in this paper is whether hierarchical clustering can be used to this end. We used different datasets and other initial layout techniques to be able to compare different approaches. We can draw the following conclusions:

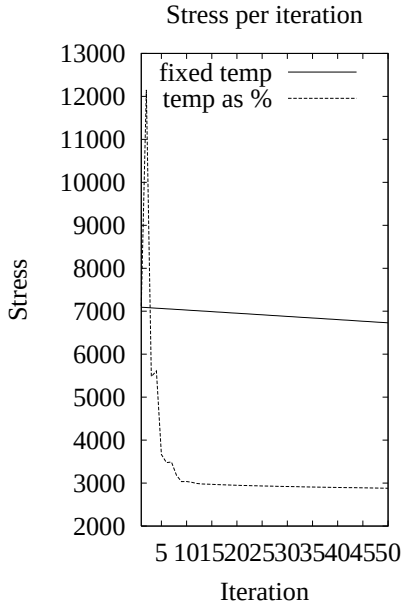


Figure 12: Improved convergence of spring embedding using temperature scheduling.

- A random initial layout performs usually very good, if temperature scheduling is used appropriately.
- The circular layout performed very poor.
- The spanning tree method performed very poor.
- Hierarchical clustering can be a very good initial layout - in fact, better than the random - if the underlying data contains clusters.
- Temperature scheduling is as important as the initial layout and a good schedule can quickly adjust any initial layout, however poor it may be.

The poor performance of the spanning tree method came as surprise and we modified it to perform better. The modification involved using the plane to layout the nodes instead of just a line as described in section 3. The results of the 2D spanning tree were an improvement, but not yet as good as the others. If one continues this approach to 3D, 4D, etc. one ultimately implements a similar transformation to PCoA. A spanning tree is also used by Tunkelang [Tun94], where the first stage of the layout consists of constructing a minimal height breadth-first spanning tree of the graph. This is related, but different from our approach. There are two other approaches to using clustering as preprocessing: Morrison, Ross, and Chalmers [MRC02] use k-means, while we use hierarchical clustering. Walshaw [Wal01] uses an iterative approach, which is very similar to ours. But there are two important differences: Both Morrison et al. and Walshaw work with typically sparse graphs, whereas we worked with complete graphs. Walshaw's approach of clustering corresponds to hierarchical clustering with single linkage. With this relationship to hierarchical clustering, one could experiment how different linkage methods influence the layout. The second difference to Walshaw is that he mainly evaluated his approach on examples, whereas we tried to go one step further and examine

examples stemming from different classes of data, such as unstructured data and hierarchically clustered data as the underlying structure of the data will greatly influence the behaviour of the algorithm.

References

[AAH94] A.Frick, A.Ludwig, and H.Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of Graph Drawing GD94*, pages 388–403. Springer, 1994.

[BETT99] G. Batista, E. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the visualisation of Graphs.* Prentice Hall, 1999.

[DEKM98] C. Durbin, S. Eddy, A Krough, and G. Mitchison. *Biological Sequence Analysis.* CUP, 1998.

[dIB97] J deRisi, V R Iyer, and P O Brown. Exploring the metabolic and genetic control of gene expression on a ge nomic scale. *Science*, 278:680–686, 1997.

[Ead84] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[ESBB98] M B Eisen, P T Spellman, P O Brown, and D Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95(25):14863–14868, 1998.

[Eve78] B. S. Everitt. *Graphical techniques for multivariate data.* Heinemann Educational Books, 1978.

[FMC96] F.J.Brandenburg, M.Himsolt, and C.Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In *Proceedings of Graph Drawing GD95*, pages 76–87. Springer, 1996.

[FR91] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software practice and experience*, 21:1129–1164, 1991.

[Gor81] A. D. Gordon. *Classification.* Chapman and Hall, 1981.

[Gow66] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53:325–38, 1966.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 1:7–15, 1989.

[Kru64] J. Kruskal. Multidimensional scaling by optimizing goodness to fit to non-metric hypotheses. *Psychometrika*, 29:1–27, 1964.

[Kru77] J. Kruskal. The relationship between multidimensional scaling and clustering. In *Classification and clustering.* Academic Press, 1977.

[MKB79] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate analysis.* Academic Press, 1979.

[MRC02] Alistair Morrison, Greg Ross, and Matthew Chalmers. Combining and comparing clustering and layout algorithms. University of Glasgow, 2002.

[Ols95] Clark F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313–1325, 1995.

[QB79] N. R. Quinn and M. A. Breuer. A force directed component placement procedure for printed cicuit boards. *IEEE Transactions on Circuits and systems*, CAS-26(6):377–388, 1979.

[RD96] R.Davidson and D.Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15:301–331, 1996.

[Sam69] J. W. Sammon. A nonlinear mapping for data analysis. *IEEE Transactions on Computers*, C(18):401–409, 1969.

[SGvHN01] Michael Schroeder, David Gilbert, Jacques van Helden, and Penny Noy. Approaches to visualisation in bioinformatics: from dendrograms to space explorer. *Information Science: An international journal*, 139(1):19–57, 2001.

[SM58] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[Tun94] D. Tunkelang. A practical approach to drawing undirected graphs. Technical report, Carnegie Mellon University, School of Computer Science, 1994.

[Tun99] D. Tunkelang. Jiggle: Java interactive graph layout enviroment. In *Proceedings of Graph Drawing GD98*, pages 413–422. Springer, 1999.

[Tut60] W. T. Tutte. Convex representations of graphs. *Proc. of London Mathematical Society*, 10:304–320, 1960.

[Tut63] W. T. Tutte. How to draw a graph. *Proc. of London Mathematical Society*, 13:743–768, 1963.

[Wal01] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. In J. Marks, editor, *Graph Drawing, 8th Intl. Symp. GD 2000*, volume 1984 of *LNCS*, pages 171–182. Springer, Berlin, 2001.

[Web99] Andrew Webb. *Statistical pattern recognition.* Arnold, 1999.